#분할 정복(divide and conquer)#13277 큰수 곱셈

a,b = map(int, input().split())  
print(a\*b)  
#왜 분할 정복 문제인지 모르겠다. 역시나 pypy3으로 시도 했을 떄 훨씬 빨랐다.

#2447 별찍기-10

**import** sys  
n = int(input())  
**def** star(i,j):  
 **while** i!=0:  
 **if** i%3 == 1 **and** j % 3 == 1:  
 sys.stdout.write(**' '**)  
 **return None** i = i//3  
 j = j//3  
 sys.stdout.write(**'\*'**)  
**for** i **in** range(n):  
 **for** j **in** range(n):  
 star(i, j)  
 sys.stdout.write(**'\n'**)

#1654 랜선 자르기

#어쨌든 답은 맞았으나 항상 그렇듯 한번에 정확하게 맞추지를 못했다.k,n = map(int, input().split())  
file = [int(input()) **for** \_ **in** range(k)]  
#의문점1. 왜 a의 초기값을 0으로 하면 오류가 날까?a,b = 1, max(file)  
ans = 0  
#의문점2. while a < b라고 하면 오류가 날까? 원래 a == b는 고려 안하는 것 아닌가?**while** a <= b:  
 mid = (a+b)//2  
 **if** mid == 0:  
 ans = 1  
 **break** add = 0  
 **for** i **in** range(k):  
 add += file[i]//mid  
 **if** add < n:  
 #의문점3. 왜 여기서 mid-1을 하는 것일까? mid로만 해도 맞을 것 같은데a,b = a,mid-1  
 **else**:  
 a,b = mid+1, b  
print(b)

#2104 부분배열 고르기(1654번과 비슷한 문제이다)

#분할 정복에는 이분 탐색과 퀵 정렬 두가지 알고리즘이 예시로 존재한다. 이 문제의 경우에는 범위의 중간을 기준으로 분할한 이후에 왼쪽, 오른쪽 결과중 답을 고른다.  
#단, 답이 기준을 포함해서 걸쳐 있을 수도 있기에 기준을 포함하는 답을 찾아 주어야 한다.  
#리스트의 오른쪽값과 왼쪽 값을 비교해서 더 큰 값을 포함하도록 리스트를 계속 확장 시켜 준다.  
#그리고 위에서 고른 분할해서 구했던 답과 비교해서 답을 갱신 시켜 준다.n = int(input())  
file = list(map(int, input().split()))  
**def** check(l, r):  
 **if** l == r:  
 **return** file[l]\*file[r]  
 mid = (l+r)//2  
 ans = max(check(mid+1, r), check(l, mid))  
  
 left, right = mid, mid+1  
 add, small = file[left]+file[right], min(file[right], file[left])  
 ans = max(ans, add\*small)  
 #아래 코드에서 계속 or 대신에 and를 사용해서 틀렸었다.**while** left > l **or** right < r:  
 **if** right < r **and** (left == l **or** file[left-1] < file[right+1]):  
 right += 1  
 add += file[right]  
 small = min(small, file[right])  
  
 **else**:  
 left -= 1  
 add += file[left]  
 small= min(small, file[left])  
 ans = max(ans, add\*small)  
 **return** ans  
  
print(check(0,n-1))

#1074 Z

n, r, c = map(int, input().split())  
**def** find(n, r, c, ans):  
 a = 2\*\*n  
 b = a//2  
 **while** n >= 1:  
 **if** n == 1:  
 **if** r == 0 **and** c == 0:  
 ans+= 0  
 **elif** r == 0 **and** c == 1:  
 ans+= 1  
 **elif** r == 1 **and** c == 0:  
 ans+= 2  
 **elif** r == 1 **and** c==1:  
 ans+= 3  
 **else**:  
 **if** 0<=r<b **and** 0<=c<b:  
 ans += 0\*(b\*\*2)  
 **elif** 0<=r<b **and** b<=c<a:  
 ans += 1\*(b\*\*2)  
 **elif** b<=r<a **and** 0<=c<b:  
 ans += 2\*(b\*\*2)  
 **elif** b<=r<a **and** b<=c<a:  
 ans += 3\*(b\*\*2)  
 **return** find(n-1, r%b, c%b, ans)  
 **return** ans  
print(find(n, r, c, 0))

#2749 피보나치 수

#처음에는 당연히 재귀적으로 풀었지만 재귀의 깊이나 데이터 적인 측면에서 초과가 조금 발생했다.  
#솔직히 웬만해서는 그냥 원래 푸는 대로 재귀함수나 dp를 이용하면 가능한데 이 문제는 n이 너무 컸다.  
#피보나치 수를 K로 나눈 나머지는 항상 주기를 가지게 된다. 이를 피사노 주기(Pisano Period)라고 한다.  
#피보나치 수를 3으로 나누었을 때, 주기의 길이는 8이다.  
#주기의 길아가 p이면 n번쨰 피보나치 수를 m으로 나눈 나머지는 n%p 번째 피보나치 수를 m을 나눈 나머지와 같다.  
#m = 10\*\*k 일때 k > 2이면 주기는 항상 15\*10\*\*(k-1)이다.n = int(input())  
p = 1500000  
n = n%p  
dp = [0]\*2  
**for** i **in** range(n+1):  
 **if** 0 < i <= 2:  
 dp[i%2] = 1  
 **else**:  
 #몰랐는데 큰 수의 덧셈의 반복만으로도 시간 초과가 발생할 수 있는 요인이 된다.dp[i%2] = ((dp[(i-1)%2]+dp[(i-2)%2]))%1000000  
print(dp[n%2])

#1725 히스토그램

#python3나 pypy3모두 시간초과가 발생하지 않는다.n = int(input())  
file = []  
**for** \_ **in** range(n):  
 file.append(int(input()))  
**def** check(l,r):  
 **if** l == r:  
 **return** file[l]  
 mid = (l+r)//2  
 left, right = mid, mid+1  
 height = min(file[left], file[right])  
 ans = height\*2  
 **while** left > l **or** right < r:  
 **if** left > l **and** (right == r **or** file[left-1] > file[right+1]):  
 left -= 1  
 height = min(height, file[left])  
 **else**:  
 right += 1  
 height = min(height, file[right])  
 ans = max(ans, height \* (right-left+1))  
 **return** max(ans, check(l, mid), check(mid+1, r))  
print(check(0,n-1))

#1822 차집합

#2. 정렬 후 이분 탐색(pypy3으로 해도 시간 초과 발생)a,b = map(int, input().split())  
file\_a = list(map(int, input().split()))  
file\_b = list(map(int, input().split()))  
file\_a.sort()  
file\_b.sort()  
ans = []  
left, right = 0, len(file\_b)  
**for** i **in** file\_a:  
 left, right = left, len(file\_b)  
 **while** left < right:  
 mid = (left+right)//2  
 **if** file\_b[mid] > i:  
 left, right = left, mid  
 **elif** file\_b[mid] < i:  
 left, right = mid+1, right  
 **elif** file\_b[mid] == i:  
 file\_a.remove(i)  
 **break**print(len(file\_a))  
**for** i **in** file\_a:  
 print(i, end = **' '**)

#1. set이용해서 직접 빼서 구하기a,b = map(int, input().split())  
file\_a = set(map(int, input().split()))  
file\_b = set(map(int, input().split()))  
left = list(file\_a-file\_b)  
print(len(left))  
left.sort()  
**for** i **in** left:  
 print(i, end = **' '**)

#5904 Moo 게임

#S(k) = S(k-1) + 'm' + 'o'\*(k+2) + S(k-1)  
#문제부터가 분할 정복이기는 하지만 실제로 문제를 풀 때에는 재귀적으로 호출하면 된다.  
#문제를 잘못 이해했었다. 문자열을 연속으로 이어 붙이는 것이 아니라 순서대로 하나의 각각의 다른 문자열로 취급 해야 하는 것이었다.  
#그래서 다시 풀어보았고, 이번에는 확실히 맍았으며 원래도 알고리즘은 맞았던 것 같다.n = int(input())  
dp = [0]\*100000  
dp[0] = 3  
order = 0  
add = 0  
**for** i **in** range(1,10000):  
 dp[i] = dp[i-1]\*2 + (i+3)  
 **if** n < dp[i]:  
 order = i  
 **break  
def** check(n,order):  
 **if** order == 0:  
 **if** n == 1:  
 print(**'m'**)  
 **else**:  
 print(**'o'**)  
 **return  
 elif** order > 0:  
 **if** dp[order-1] < n < dp[order-1] + (order+4):  
 n-= dp[order-1]  
 **if** n == 1:  
 print(**'m'**)  
 **else**:  
 print(**'o'**)  
 **return  
 elif** n <= dp[order-1]:  
 check(n,order-1)  
 **elif** n >= dp[order-1] + order+4:  
 n -= (dp[order-1]+order+3)  
 check(n,order-1)  
check(n,order)

**대표적인 정렬 알고리즘 4가지**

|  |  |
| --- | --- |
| 1. 병합 정렬(O(nlogn)) | 2. 삽입 정렬(O(n^2)) |
| 재귀적으로 병합 정렬을 이용해서 리스트를 정렬한다. | 배열의 각각 요소들을 앞에서부터 차례로 비교해 나간다. 앞에 있는 수열은 이미 정렬 되었을 것이기 때문에 하나하나 비교하다가 더 작은 수가 나온다면 그때 멈추고 더 작은 수의 뒤에 배치 |
| def sort(x):  if len(x) < 1:  return x  left = sort(x[:len(x)//2])  right = sort(x[len(x)//2:])  i,j,k = 0,0,0  while i < len(left) and j < len(right):  if left[i] < right[j]:  x[k] = left[i]  i += 1  else:  x[k] = right[j]  j += 1  k += 1  if i < len(left):  while i < len(left):  x[k] = left[i]  i += 1  k += 1  elif j < len(right):  while j < len(right):  j += 1  k += 1  return x | **def** sort(x):  **for** i **in** range(1, len(x)):  j,k = i-1, x[i]  **while** x[j] > k **and** j >= 0:  x[j+1] = x[j]  j -= 1  x[j+1] = k  **return** x |
| 3. 버블 정렬(O(n^2)) | **4. 퀵 정렬(O(nlogn) ~ O(n^2))** |
| 구현은 쉽지만 시간 복잡도가 너무 크다. 인접한 두 원소를 비교하여 스왑하는 것을 반복하는 알고리즘이다. | 시간 복잡도가 제일 작은 정렬 알고리즘이다.  1. 전체에서 기준이 되는 원소 하나 정하기  2. 기준으로 왼쪽에는 기준보다 작은 수, 오른쪽에는 기준보다 큰 수  3. 기준을 중심으로 각각 왼쪽, 오른쪽 리스트에서 기준을 또 정해서 정렬 반복 |
| def sort(x):  for i in range(len(x)-1):  for j in range(len(x)-i):  if j+1 < len(x):  if x[j] > x[j+1]:  x[j], x[j+1] = x[j+1], x[j]  return x | **def** sort(x):  **if** len(x) <= 1:  **return** x  mid = x[len(x)//2]  left, right, same = [],[],[]  **for** i **in** range(len(x)):  **if** x[i] < mid:  left.append(x[i])  **elif** x[i] > mid:  right.append(x[i])  **else**:  same.append(x[i])  **return**(sort(left)+same+sort(right)) print(sort(x)) |